AMENDMENTS TO THE SPECIFICATION

Please replace the following paragraphs no. 86, 87, 90, 133, 138, 139, 140, 141, 143-147 with the following amended paragraph:

[86] (a) SAT proof analysis techniques which use CNF (Conjunctive Normal Form) representation for the given Boolean SAT problem is extended to work with a hybrid SAT solver. A hybrid SAT solver uses hybrid representations of Boolean constraints, e.g. where a circuit netlist is used to represent the original circuit problem, and CNF is used to represent the learned constraints. An example implementation for a SAT solver with Proof Analysis is shown in FIG. 3. Given a SAT problem, not in CNF form (box 300), we start with preprocessing (box 301). If this generates a conflict (box 302), conflict analysis is performed (box 303). This is followed by marking all the reasons for implications which lead to the conflict (box 304). Note that since the reasons are themselves constraints, they have hybrid representations in a hybrid SAT solver.

[87] If, due to the conflict, the problem is unsatisfiable, we terminate (box 305). If there is no conflict found during preprocessing, we proceed to check if any more decisions are needed on variables (box 310). If not, then the problem is found to be satisfiable, and we terminate (box 311). Otherwise, we pick a decision variable and its value using some heuristics, and perform Boolean Constraint Propagation (BCP) (box 320). Next, we check if there is any conflict (box 321). If not, then we loop back to box 310.

[90] Next, we follow the procedure as described in (a) above (boxes 403-405, boxes 407, 410, 411, 421, 430, 431, 432, 433, 440, and 441 are similar to boxes 305, 310, 311, 321, 330, 331, 332, 333, 340, and 341 respectively). If the problem is unsatisfiable, then a recursive marking procedure is invoked (boxes 406, and 440). In this procedure, reasons for all learned constraints, including those derived during Boolean simplification, are marked recursively. Again, the marked reasons from the original problem constitute a sufficient set of reasons for unsatisfiability in the presence of Boolean simplification.

[133] (a) We use proof analysis techniques (described above) with BMC Search for generation of abstract models. Our use of BMC Search with Proof Analysis is shown in FIG. 5. Given a model and a safety property, we start at depth 1 (box 500). If we have not reached the maximum depth (boxes 501, 502) specified (k_{max}), then we formulate the k-instance BMC problem (box 503). Next, we use the SAT solver with Proof Analysis on this problem (box 504). If the problem is satisfiable (box 503), then a counterexample has been found (corresponding to the satisfying assignment of the SAT problem) and we terminate (box 506). However, if the problem is unsatisfiable, i.e. no counterexample of length k exists, then we save the set of reasons for the unsatisfiability, marked by the proof analysis technique, and associate this set with depth k (box 507). Next, we increment k (box 508). If we have reached the limit k_{max}, we stop (box 510). At this point, we have k sets of reasons, one for each depth up to k. These reasons are used to extract abstract models, one for each depth up to k (box 509).

[138] (a) We use a novel iterative abstraction framework based on use of BMC Search with Proof Analysis. This is shown in FIG. 8. Given a concrete design and a correctness property (box 800), we start with iteration index n = 1 (box 801). The model for the first iteration, A_1, is chosen to be the given concrete design (box 802). In general, in the n-th iteration, we use BMC Search with Proof Analysis on the model A_n (box 803). If a counterexample is found (box 804) during this search at depth d, we generate a new model $A_{n'}$ for the same iteration (box 810), as described in 8 (b). However, if no counterexample is found up to some specified k_{max} (potentially different for each iteration) we heuristically choose a sufficient set of reasons accumulated up to a certain depth d from 1 to k_{max} (box 820). For example, we can choose a set that remains unchanged for a certain number of time frames. Next, we extract the accumulated sufficient abstract model for depth d (box 821), as described in 7(d). This is used as the model A_{n+1} for the next iteration. If we wish to continue (box 822) the abstraction flow, we increment the iteration index n (box 823), and loop back to perform the next iteration (box 803). If we don't wish to continue, e.g. if resources for this iteration have been exhausted, or if model A_{n+1} is unchanged from model A_n, we attempt to verify the abstract model A_{n+1} (box 830).

[139] We use various methods for this verification, described later in (d)-(h). If the verification finds a counterexample (box 831) for model A_{n+1} at depth d, then we handle this counterexample (box 832), as described in 8 (b). If no counterexample was found, and a conclusive result was obtained (check in box 840), then we can

stop (box 841) with the conclusive result holding for the concrete design. If the result was inconclusive, e.g. if verification could not be completed, then we can either (1) choose a different set of reasons at a depth d' which is greater than the depth d from which model A_{n+1} was extracted (box 842), provided d' is less than k_{max} of this iteration; or (2) we can choose to perform the next iteration by incrementing the iteration index n (box 823). The loop is guaranteed to terminate due to the finite concrete design, and in the worst case BMC may need to run up to the longest possible path in the concrete design.

[140] (b) Given an abstract counterexample found on an abstract model A_n at depth d (box 900), we use the procedure shown in FIG. 9 to generate a new model $A_{n'}$ for the same iteration of the iterative abstraction flow. We first run BMC Search with Proof Anslysis on model A_{n-1} from the previous iteration up to some depth d' > d (if this hasn't been done already) (box 901).. If BMC Search can be completed for some depth d' > d (box 902), we extract ASM(d') use it as the new model $A_{n'}$ (box 903). Otherwise, we use a refinement procedure described in 8(c) to obtain a new model $A_{n'}$ (box 904). In either case, we re-enter the Iterative Abstraction Flow with the new model $A_{n'}$ (box 905).

[141] (c) The proof analysis-based refinement procedure works as shown in FIG. 10. Given a counterexample on model A_n at depth d (box 1000), we first generate Boolean constraints corresponding to the counterexample, separately for each time frame up to d (box 1001). The Refinement with Proof Analysis procedure starts from time frame k=1 (box 1002). We formulate the SAT problem for checking constraints at time frame k on the concrete design (box 1003). Next, we use the

SAT solver with Proof Analysis on this problem (box 1004). If the problem is unsatisfiable (box 1005), then the abstract counterexample is spurious at time frame k. We use the reasons for the unsatisfiability at time frame k, to refine the model A_n to $A_{n'}$ (box 1006 and 1007). Specifically, we use the pruned set of sufficient constraints, rather than the entire set R(k), to identify suitable refinement candidates. Refinement is targeted at adding latches to the model A_n, in order to remove the spurious counterexample. However, if the problem is satisfiable, then we increment index k (box 1008). If k is greater than d (box 1009), then we have found a true counterexample on the concrete design, and we stop (box 1010). We actually quit also from the surrounding iterative abstraction flow. Otherwise, we loop back to checking constraints at the new k (box 1003).

[143] (d) We use BMC Search technique to search for a counterexample on the abstract model A_n, as shown in FIG. 11 (boxes 1100-1102). If there is no counterexample up to depth k (box 1103), then it is guaranteed that there is no counterexample up to depth k in the concrete design either, i.e. the concrete design is correct up to depth k. In many cases, BMC Search can do deeper searches on the smaller abstract models than on the larger concrete design. However, if a counterexample is found (box 1104), then we can handle it as described in 8(b).

[144] (e) For safety properties, we use BMC Proof technique [5] to perform a proof by induction with increasing depth on the abstract model A_n, as shown in FIG. 12 (boxes 1200-1202). If the proof by induction succeeds (box 1203), then it is guaranteed to be true on the concrete design also. In many cases, a BMC Proof technique may be able to perform a deeper proof by induction on an abstract

model, which may allow it to succeed. If the proof fails up to the maximum depth tried (box 1204), the verification is inconclusive.

[145] (f) We use BDD-based [7, 8] or SAT+BDD-based [9] symbolic model checking methods on the abstract model A_n, as shown in FIG. 13 (boxes 1300-1302). If the correctness property is proved to be true (box 1303), it is guaranteed to be true on the concrete design as well. Due to the limited capacity of such methods, they are more likely to work on smaller abstract models. If the property fails, an abstract counterexample is found (box 1304), and we can handle it as described in 8(b).

[146] (g) For safety properties, we use BDD-based [7, 8] or SAT+BDD-based [9] reachability analysis methods on an abstract model A_n to constraint BMC Proof on the concrete design (or any abstract model larger than A_n), as shown in FIG. 14. Given an abstract model A_n (box 1400), we use approximate or exact traversal methods to obtain BDDs that denote sets of reachable states at different depths (box 1401). These sets are over-approximations of reachable states in the concrete design. Next, we convert the reachable set BDDs into Boolean constraints (box 1402). Then we use BMC Proof technique [5] on the concrete design, where the reachable set constraint is used to constrain the arbitrary state in the inductive step (box 1410). If the proof by induction succeeds (box 1411), then it is guaranteed to be true on the concrete design also (box 1412). In many cases, the use of a BDD-based reachability invariant allows the induction proof to succeed. If the proof fails, verification is inconclusive (box 1413).

[147] (h) We use BDD-based [7, 8] or SAT+BDD-based [9] reachability analysis methods on an abstract model A_n to constrain BMC Search for counterexamples on the concrete design (or any abstract model larger than A_n), also shown in FIG. 14. The procedure for deriving the reachability constraints is the same as that described in 8(g) (boxes 1401, 1402). Then we use BMC Search, where reachability constraints provide additional constraints (box 1420). In practice, use of reachability constraints with BMC Search can help to perform deeper searches. If no counterexample is found (box 1421) up to depth k (box 1422), then the concrete design is guaranteed to be correct up to depth k. However, if a counterexample is found (box 1423), we handle it as described in 8(b).